Tractable query languages for complex object databases

Stéphane Grumbach, Victor Vianu

UNITÉ DE RECHERCHE
INRIA-ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1573

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

TRACTABLE QUERY LANGUAGES FOR COMPLEX OBJECT DATABASES

Stéphane GRUMBACH
Victor VIANU

Décembre 1991

Langages de requêtes polynômiaux pour objets complexes

Stéphane Grumbach
I.N.R.I.A.
Rocquencourt BP 105
78153 Le Chesnay, France
grumbach@seti.inria.fr

Victor Vianu
CSE C-0114
UC San Diego
La Jolla, CA 92093, USA
vianu@cs.ucsd.edu

Abstract

Nous étudions le pouvoir d'expression et la complexité de différents langages de requêtes pour objets complexes. Contrairement aux travaux précédents, nous étudions la complexité des requêtes sur des bases de données contenant des objets complexes plutôt que des bases de données plates. Ceci soulève de nouveaux problèmes propres aux objets complexes. Nous montrons par exemple que la manière dont la base de données utilise les types d'ordre supérieur a un impact direct sur la complexité. L'utilisation d'opérateurs de point fixe donne des langages qui se comportent bien quant à la complexité et le pouvoir d'expression. En particulier, nous montrons qu'une extension des requêtes point fixe aux objets complexes exprime exactement les requêtes PTIME sous l'hypothèse que la base de données utilise pleinement tous ses types. Des résultats similaires sont basés sur des requêtes à domaine restreint.

Tractable Query Languages for Complex Object Databases*

Stéphane Grumbach
I.N.R.I.A.
Rocquencourt BP 105
78153 Le Chesnay, France
grumbach@seti.inria.fr

Victor Vianu
CSE C-0114
UC San Diego
La Jolla, CA 92093, USA
vianu@cs.ucsd.edu

October 31, 1991

Abstract

The expressiveness and complexity of several calculus-based query languages for complex objects is considered. Unlike previous investigations, we are concerned with the complexity of queries on databases of complex objects, rather than flat databases. This raises new issues specific to complex objects. For instance, it is shown that the way the database makes use of its higher-order types has direct impact on query complexity. The use of fixpoint operators is shown to yield languages well-behaved with respect to complexity and expressiveness. In particular, an extension of the *fixpoint* queries to complex objects is shown to express precisely the PTIME queries, under the assumption that the database makes "full" use of all its types. Similar results involve range-restricted queries.

1 Introduction

Complex objects are increasingly part of advanced database systems. They provide the structural core of object-oriented databases. Several query languages for complex objects have been proposed. However, the complexity and expressiveness of such query languages remains little understood. In this paper we investigate the expressiveness and complexity of several query languages for complex object databases. The results provide new tools for evaluating the complexity of queries on such databases and suggest constructs for tractable query languages.

*Work supported in part by an INRIA-NSF cooperation grant, and the National Science Foundation under grants IRI-8816078 and INT-8817874.

Languages for complex objects which have been proposed are roughly of three types: extensions of relational calculus [Jac82, AB87, KV84, KRS85], extensions of relational algebra [AB87, AB86, FT83, KV84, SS86] and deductive languages [AG91, BNR+87, Kup87, Kup88]. They all use higher-order types. Previous investigations of their expressive power have focused on the gain in expressivity resulting from the use of higher-order types, when queries are applied to *flat* databases [HS88, KV88]. In this case, the use of higher-order types results in very high complexity with respect to the flat input.

This paper focuses on queries whose inputs are *complex object* databases, rather than flat databases. We aim to provide query languages for complex object databases, whose complexity is similar to that of traditional query languages, like the relational calculus or its recursive extensions, the *fixpoint* queries and the *while* queries [CH80]. We primarily use an extension of relational calculus to complex objects. With an eye to tractability, we focus on restrictions of the calculus which use types of the same set height and arity as those of objects in the input database. We also use fixpoint operators in conjunction with these languages. These are redundant in the context of unrestricted higher-order types, and therefore have not been included in previously proposed query languages for complex objects. However, we argue that fixpoint operators provide a tractable form of recursion, and show that they yield languages which are well-behaved with respect to expressive power. The fixpoint extensions of the calculi are closely related to deductive languages like Datalog extensions to complex objects.

As mentioned above, we attempt to limit the complexity of queries by requiring that the types used in the query be similar to those of objects in the database. However, the effectiveness of this restriction depends on the way the database makes use of its types. Both the database and the queries may use higher-order types. However, if the database contains mostly atomic values and few objects of higher-order types, then the query may still have high complexity with respect to the database. On the other hand, if the database is large relative to the domains of higher-order types, the complexity of a query may be low, relative to the size of the database, although it involves such types. This gives rise to the notions of "sparsity" and "density" of the set of valid database instances with respect to given types. Intuitively, density of a database with respect to a given type indicates that the database makes full use of that type, so its size is comparable to that of the domain of the type; sparsity with respect to a type indicates that the use of the type by the database is just cosmetic. We claim that information on density or sparsity of the database is often available; it can be thought of as a form of integrity constraint. Such information can be used to evaluate the complexity of queries on the database. Indeed, we provide characterizations of the expressive power of various languages under sparsity and density assumptions on the database. In particular, we exhibit a language which expresses exactly the PTIME queries, provided the database is dense with respect to its types. Note that, unlike the flat case, no additional assumption, like an order on the constants, is required. This is essentially a consequence of the fact that a dense database is large relative to the number of atomic constants. The main interest of the results with sparsity and density assumptions is that they highlight the interplay between type usage by the database and query complexity. This suggests new ways of tuning query complexity and expressiveness given information on type usage.

A second way to limit the gap between a query and the input database is by a syntactic restriction called "range-restriction", which limits the range of variables in the query. It is close to other notions of range-restriction previously defined in the literature [AB87, AG91, AK89, BNR+87, Kup87]. We obtain expressiveness results for range-restricted queries, which are similar to those obtained with density and sparsity assumptions.

In order to provide a link with existing results, we consider the expressive power of the languages with respect to the queries on flat databases. The fixpoint extensions of the calculi are shown to have precise characterizations in terms of complexity, unlike the previously studied calculi without the fixpoint operators [HS88, KV88].

The paper consists of seven sections. In the next section we develop the framework for complex objects, queries and complexity classes of queries on complex objects. This extends the classical framework of Chandra and Harel [CH80] for flat queries. Section 3 reviews calculus-based query languages for complex objects, and introduces the new fixpoint extensions. The expressiveness results with sparsity and density assumptions are presented in Section 4. Section 5 is devoted to range-restricted queries. The flat case is considered in Section 6. Finally, some conclusions are provided in Section 7.

2 Complex Objects and Queries

In this section, we review the main definitions concerning complex objects, and set the basic framework for the study of queries on complex object databases.

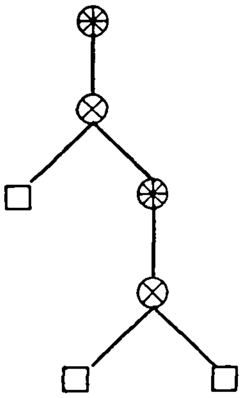
Complex objects are typed objects, built recursively from atomic objects (constants) using set and tuple constructors. We assume the existence of a set constructor { }, independent of the cardinality of the sets, and of k-ary tuple constructors [ ] for every positive integer k. The types and their domains are defined recursively as follows. Let U be an *atomic type*, and $dom(U)$ an infinite set of atomic constants. We define next the complex types and their domains with respect to a given subset D of $dom(U)$.

1. U is the *atomic* type, and $dom(U, D) = D$.

2. If T is a type, $\{T\}$ is a *set type* with domain : $dom(\{T\}, D) = \wp_{fin}(dom(T, D))$, where $\wp_{fin}(S)$ denotes the set of finite subsets of S.

3. If $T_1, ..., T_n$ are types, $[T_1, ..., T_n]$ is a *tuple type* with domain : $dom([T_1, ..., T_n], D) = \{[a_1, ..., a_n]/a_i \in dom(T_i, D)\}$.

We will sometime write $dom(T)$ instead of $dom(T, D)$ when either D is understood from the context, or when D is irrelevant. The set of atomic constants occurring in an object O is denoted by $atom(O)$.



The type $\{[U,\{[U,U]\}\}$

A type can be represented as a labeled tree, where the leafs are labeled by $\square$ and the internal nodes are labeled with the set ($\circledast$) or tuple ($\otimes$) constructor. For instance, the type $\{[U,\{[U,U]\}]\}$ is represented in the figure. The *set height* of a type T is the maximum number of set nodes in a path from the root to a leaf in T. The *tuple width* of a type T is the maximal tuple width (number of arguments) of the tuples occurring in T. The type $\{[U,\{[U,U]\}\}$ has set height 2 and tuple width 2. The notation and terminology are from [AH87].

We will use restrictions on types based on their set height and tuple width. An $< i,k >$-*type* is a type whose set height is at most i and whose tuple width is at most k. For an $< i,k >$-type T, the cardinality of $dom(T,D)$, where the cardinality of D is $|D| = n$, is bounded by the hyperexponential function $hyper(i,k)$ in n, defined by :

$$hyper(i,k)(n) = 2^{k2^{\cdot^{\cdot^{\cdot 2^{kn^k}}}}} \}i$$

where the number of occurrences of "2" in the tower of exponentials is equal to i. The domain of the $< i,k >$-types with respect to a given set D of constants is defined by :

$$dom(i,k,D) = \bigcup\{dom(T,D)/T \text{ is an } < i,k >\text{-type}\}.$$

A type T is *non trivial* if $i \geq 1$ and $k \geq 2$, i.e. if the set and tuple constructors are used at least one in a non trivial way. This distinction is important for technical reasons which will become clear later.

A database of complex objects is an extension of a relational database. It consists of relations among complex objects of specified types. A *relation schema* is an expression $R[T_1,...,T_n]$, where R is a *relation name* and $T_1,...,T_n$ are types. An *instance* of $R[T_1,...,T_n]$ is a finite subset of $dom(T_1) \times ... \times dom(T_n)$. A *database schema* **R** is a finite set of relation schemas with distinct relation names. An *instance* over a database schema **R** is a mapping associating to every relation schema in **R** an instance of that relation schema. Instances are denoted **I**,**J**,**K**,.... The set of instances over **R** is denoted **inst(R)**.

An $< i,k >$-*database schema* **R** is a database schema such that, for each $R[T_1,...,T_n]$ in **R**, each T_j, $(j = 1..n)$, is an $< i,k >$-type. Note that no restriction is placed on the arity of the relations of an $< i,k >$-database schema.

Queries on complex object databases are defined by extending the classical definition of [CH80] for flat queries. A query is a mapping from an input schema $\mathbf{R} = \{R_1,...,R_n\}$ to an output schema $\mathbf{S} = \{S\}$ with a single relation, mapping instances of the input schema to instances of the output schema. Queries must be computable, and insensitive to isomorphisms on the atomic constants. Complexity classes of queries on complex objects are

P		
b	$\{a,b\}$	$[c,\{a,c\}]$
c	$\{c\}$	$[a,\{b,c\}]$

Figure 1: An instance **I**.

defined straightforwardly by extending the definition for flat queries. We use as complexity measures the time (space) used by a Turing machine to produce a standard encoding of the output database starting from a standard encoding of the input database. The measures are functions of the size of the input database instance. Note that the complexity of a query is not expressed in terms of the corresponding recognition problem but rather in terms of the complexity of computing the result (unless stated otherwise). Thus, for each Turing complexity class C there is a corresponding complexity class of queries, which, for simplicity, we also denote by C. For instance, we will refer to the PTIME and PSPACE queries. If F is a set of functions, P(F)-time (-space) denotes the complexity class of queries for which time (space) is bounded by some polynomial in some function $f \in F$ applied to the size of the input.

We brifly dwell on the standard encoding of database instances on Turing tapes. These will be required in several of our proofs. The type constructor symbols $\{,\},[,]$, as well as the relation names, are part of the tape alphabet. The symbol # is used as a separator. We begin with the encoding of a complex object of some type T over a finite domain D of n atomic constants. Each of the n constants is encoded in binary notation, using $log(n)$ space. More precisely, suppose $a_1, ..., a_n$ is an arbitrary enumeration of D. Then the encoding of a_i, denoted $enc(a_i)$, is the binary representation of the integer i. Let $o = [t_1, ..., t_k]$ be an object in $dom(T, D)$, where $T = [T_1, ..., T_k]$. Then $enc(o) =$ "$[enc(t_1)\#...\#enc(t_k)]$". Next, suppose $o \in dom(T, D)$, where $T = \{T'\}$. Let $\{t_1, ..., t_k\}$ be an enumeration of the objects in o, in the natural order[1] induced on objects of type T' by the enumeration $a_1, ..., a_n$ of D. Then $enc(o) =$ "$\{enc(t_1)\#...\#enc(t_k)\}$". The encoding of an instance **I** of a relation P is done similarily. Let $t_1, ..., t_k$ be an enumeration of the tuples in **I**, once again in the order induced by the initial enumaration of D. Then $enc(\mathbf{I}) =$ "$Penc(t_1)...enc(t_k)$". If **R** consists of several relation schemas $R_1, ..., R_k$, $enc(\mathbf{I}) =$ "$enc(\mathbf{I}(R_1))\#...\#enc(\mathbf{I}(R_k))$".

Note that, strictly speaking, there are several standard encodings of a given instance **I**, one for each enumeration of its atomic constants.

Example 2.1 Let P be a ternary relation of type $[T_1, T_2, T_3]$, where $T_1 = U$, $T_2 = \{U\}$, and $T_3 = [U, \{U\}]$. Let **I** be the instance of P represented in Figure 1. Let $D = \{a, b, c\}$ and abc be an enumeration of the constants in D. Then $enc(\mathbf{I})$ is represented in Figure 2.

The *size* of a database G is the size of its standard encoding, and is denoted by $||G||$. The cardinality of a database G, denoted by $|G|$, is the sum of the cardinalities (number of

[1] See Definition 4.2 for a detailed definition of the induced order.

P[01#{00#01}#[10#{00#10}]][10#{10}#[00#{01#10}]]

Figure 2: Encoding of **I** on a TM tape

tuples) of its relations. Note that the distinction between the size of an input relation over complex objects and its cardinality is relevant, unlike the flat case. For instance, a unary relation of cardinality 1 may contain a single very large object and thus have arbitrarily large size. However, the difference between size and cardinality is less dramatic for *domains* of types, as shown in the next proposition.

Proposition 2.1 For each type T there exists a polynomial P such that, for each finite set of constants D,

$$||dom(T,D)|| \leq |dom(T,D)| P(log(|dom(T,D)|)).$$

Proof : Let D be a set containing n constants. The proof is by induction on the imbrication of constructors in T (we make the usual assumption that there are no tuple constructors imbricated directly into other tuple constructors, i.e. there is always a set constructor between two imbricated tuples).

First, if $T = U$, it is easy to check that : $||dom(U,D)|| = k|dom(U,D)|log(|dom(U,D)|)$. for some constant k. If $T = \{T'\}$ then $enc(dom(T,D))$ consists of the concatenation of $2^{|dom(T',D)|}$ encodings of subsets of $dom(T',D)$, each of which takes at most $||dom(T',D)||$ space. Thus,

$$\begin{aligned}
||dom(T,D)|| &\leq c\, 2^{|dom(T',D)|}\, ||dom(T',D)|| \text{ (for some constant c)}\\
&= c\, |dom(T,D)|\, ||dom(T',D)||\\
&\leq c\, |dom(T,D)|\, |dom(T',D)|\, P(log(|dom(T',D)|))\\
&= c\, |dom(T,D)|\, log(|dom(T,D)|)\, P(log(log(|dom(T,D)|)))\\
&\leq |dom(T,D)|\, Q(log(|dom(T,D)|)) \text{ (for some polynomial } Q).
\end{aligned}$$

Next, suppose $T = [T_1, ..., T_k]$. Then $enc(dom(T,D))$ consists of the concatenation of $\prod_{i=1}^{i=k} |dom(T_i,D)|$ encodings of tuples of the form $[o_1, ..., o_k]$, where each o_i is in $dom(T_i,D)$. Note that $\prod_{i=1}^{i=k} |dom(T_i,D)| = |dom(T,D)|$. Also, for each T_i, there are two cases: $T_i = U$, or $T_i = \{T_i'\}$ for some T_i'. In the first case, $||enc(o_i)|| = log(|D|) \leq log(|dom(T,D)|)$. In the second case, $||enc(o_i)|| \leq ||dom(T_i',D)||$ and, by the induction hypothesis, $||dom(T_i',D)|| \leq |dom(T_i',D)|P_i(log(|dom(T_i',D)|))$ for some polynomial P_i. Also, $|dom(T_i',D)| \leq log(|dom(T,D)|)$. It follows that in all cases $||enc(o_i)|| \leq Q_i(log(|dom(T,D)|))$ for some polynomial Q_i. Thus, $||dom(T,D)|| \leq |dom(T,D)| \prod_{i=1}^{i=k} Q_i(log(|dom(T,D)|)) \leq |dom(T,D)|Q(log(|dom(T,D)|))$ for some polynomial Q. □

3 Query Languages

Many existing languages for complex objects are typed extensions of the flat relational calculus, with the constructible types defined earlier. Such languages were first defined in [KV84], based on an earlier proposal of [Jac82]. Other variations have been defined in

[AB87, HS88, KRS85]. The particular extension we use here, CALC, is from [HS88]. We briefly recall its syntax and semantics.

CALC is a strongly typed extension of first order logic using the constructible types defined above. The alphabet includes typed logical predicates for each type T, *equality* $=_T$, *membership* $\in_T$ and *containment* $\subseteq_{\{T\}}$ to manipulate sets. We will generally omit the type indices. It also provides, for each positive integer i, functions $.i$ which associate to a tuple of arity at least equal to i its i^{th} component.

Terms in CALC consist of the following: (i) complex objects of some type T, (ii) variables whose types can be inferred from the context, and (iii) expressions $x.i$ where x is a variable of type $[T_1, ..., T_n]$ and $i \leq n$.

Atomic formulas are typed expressions of the form $t_1 = t_2$, $t_1 \in t_2$, $t_1 \subseteq t_2$ and $R(t_1, ..., t_n)$, where R is a relation name, with the obvious type compatibility restrictions.

Formulas are constructed as usual using the logical connectives $\wedge, \vee, \neg, \rightarrow$ and typed quantifications $\forall x : T$ and $\exists x : T$. Note that the types of the free variables in a formula can be inferred. However, they are usually made explicit for convenience. The set of types of a formula ϕ consists of the types of terms occurring in ϕ. Well-formed formulas are defined with the obvious restrictions. The complete definition is provided in [HS88].

Finally, a *query* from an input schema $\mathbf{R} = \{R_1, ..., R_n\}$ to an output schema $\mathrm{S} = \{S[T_1, ..., T_k]\}$, is an expression of the form $\{[x_1, ..., x_k] : [T_1, ..., T_k]/\phi(x_1, ..., x_k)\}$, where ϕ is a formula whose relations are in the input schema and containing the free variables $x_1, ..., x_k$ of types $T_1, ..., T_k$.

For example, the following CALC formula defines a query on a graph G whose nodes are objects of type U, and whose answer is a graph of the same type as the input. If the answer is the graph itself, then the graph is bipartite. Otherwise, the answer is empty.

$$\{t : [U, U] \;/\; G(t) \wedge \exists X : \{U\} \; \exists Y : \{U\} \; (\neg \exists n : U(n \in X \wedge n \in Y)$$
$$\wedge \;\; \forall v : [U, U](G(v) \Rightarrow ((v.1 \in X \wedge v.2 \in Y) \vee \; (v.1 \in Y \wedge v.2 \in X))))\}.$$

The semantics is based on the active domain interpretation. Consider a formula ϕ over a database schema $\mathbf{R}$ and $\mathbf{I}$ an instance of $\mathbf{R}$. Let $atom(\mathbf{I})$ denote the set of atomic constants occurring in $\mathbf{I}$. Then in the evaluation of ϕ on $\mathbf{I}$, each variable of type T in ϕ ranges over $dom(T, atom(\mathbf{I}))$. The definition of satisfaction of a sentence by an instance is the usual one. For example, $\mathbf{I} \models \forall x : T\phi(x)$ if for every object a in $dom(T, atom(\mathbf{I}))$, $\mathbf{I} \models \phi(a)$. If Q is a query defined by a formula ϕ over an input schema $\mathbf{R} = \{R_1, ..., R_n\}$ to an output schema $\mathrm{S} = \{S[T_1, ..., T_k]\}$ and $\mathbf{I}$ is an instance of $\mathbf{R}$, then the answer of Q on $\mathbf{I}$ is :

$$Q(\mathbf{I}) = \{[x_1, ..., x_k] : [T_1, ..., T_k]/\mathbf{I} \models \phi(x_1, ..., x_k)\}.$$

In [HS88], queries from *flat* databases to flat answers, where the set height of intermediate types used in the query is restricted but not necessarily flat, are considered. The use of nested sets results in queries whose complexity is very high with respect to the flat inputs. In this paper we also consider queries with restricted types, but do so in search for tractable query languages for complex object databases. To this end, we consider queries which manipulate objects of types comparable to those in the input database and the answer. The restrictions used here are based on the set height and tuple width of the types of the formulas. The subset of CALC restricted to formulas whose types are $< i, k >$-types is denoted CALC_i^k. Note that, in particular, this implies that the input database and answer also consist of relations over objects of $< i, k >$-types.

Even with such restrictions, it turns out that the calculus is not well-behaved with respect to expressiveness of complexity classes. This is due in part to the fact that recursion involving types of set height i is expressed using types of set height $i+1$. This has a high complexity cost (exponential) with respect to inputs of set height i. We propose a more subtle form of recursion over inputs of set height i, which remains polynomial with respect to the input. It involves an extension of the flat *fixpoint* and *while* queries to complex objects, using fixpoint operators. Indeed, the results in this paper suggest that such fixpoint operators play an important role with respect to complexity and expressiveness.

The fixpoint operators added to CALC_i^k allow defining inductively new relations, by iterating CALC_i^k formulas. We define two fixpoint operators: *IFP* (inflationary) and *PFP* (partial). The difference between the *IFP* and *PFP* operators lies in the way the CALC_i^k formulas are iterated. For the *IFP* operator, the iteration results in an increasing sequence of relations, which guarantees convergence. For the *PFP* operator, the iteration is generally not increasing, and convergence is no longer guaranteed.

The fixpoint operators defined here are extensions of the *IFP* and *PFP* operators previously defined in conjunction with the first-order logic *FO* (flat relational calculus). The languages $FO+IFP$ and $FO+PFP$ have been shown to define the *fixpoint* [GS85] and *while* queries [AV89], respectively. The *fixpoint* queries are particularly important: they express exactly the PTIME queries, in the presence of an order on the domains [Imm86, Var82]. The *while* queries express the PSPACE queries in the presence of order [Var82].

The fixpoint extensions of CALC_i^k, $\mathrm{CALC}_i^k + IFP$ and $\mathrm{CALC}_i^k + PFP$ are defined next.

Definition 3.1 Let $\mathbf{R}$ be a database schema. $\mathrm{CALC}_i^k + IFP$ formulas over $\mathbf{R}$ are obtained by repeated applications of the CALC_i^k operators and the inflationary fixpoint operator *IFP*. Let $S[T_1, ..., T_n]$ be a typed predicate where each T_j, $j = 1..n$, is an $< i, k >$-type, and S is not a relation in the database schema $\mathbf{R}$. Let $\phi(S)$ be a $\mathrm{CALC}_i^k + IFP$ formula with n free variables $x_1 : T_1, ..., x_n : T_n$, involving S. Then, for a given instance over the relations in $\phi(S)$ other than S, $IFP(\phi(S), S)$ denotes the relation of type $[T_1, ..., T_n]$ which is the limit of the sequence defined by: $J_0 = \emptyset$ and for each $i > 0$, $J_i = \phi(J_{i-1}) \cup J_{i-1}$. The expression $IFP(\phi(S), S)$ can be used as a term or as a predicate. Thus, if x is a variable of type $\{[T_1, ..., T_n]\}$, $x = IFP(\phi(S), S)$ is a $\mathrm{CALC}_i^k + IFP$ formula. And, if $x_1, ..., x_n$ is the sequence of distinct free variables of $\phi(S)$, $IFP(\phi(S), S)(x_1, ..., x_n)$ is also a $\mathrm{CALC}_i^k + IFP$ formula[2]. The definition of $\mathrm{CALC}_i^k + PFP$ formulas is the same except for the semantics of the fixpoint operator *PFP*: $PFP(\phi(S), S)$ denotes the relation of type $[T_1, ..., T_n]$ which is the limit, *if it exists*, of the sequence defined by $J_0 = \emptyset$ and for each $i > 0$, $J_i = \phi(J_{i-1})$.

[2]Note that other choices are possible, such as having the arguments $x_1, ..., x_n$ be independent of the free variables of $\phi(S)$. Such variations do not affect the power of the language. We made this particular choice for technical reasons relating to range restriction.

Example 3.1 Let G be a binary relation over types $[\{U\},\{U\}]$ and consider the formula:

$$\phi(S) = G(x,y) \ \vee \ \exists\, z : U\ (S(x,z) \wedge G(z,y)).$$

The following $\mathrm{CALC}_1^1 + \mathit{IFP}$ query defines the transitive closure of G:

$$\{(x : \{U\}, y : \{U\}) \mid \mathit{IFP}(\phi(S), S)(x,y)\}.$$

Alternately, the following is a $\mathrm{CALC}_2^2 + \mathit{IFP}$ query defining the transitive closure of G:

$$\{x : \{[\{U\},\{U\}]\} \mid x = \mathit{IFP}(\phi(S), S)\}.$$

The nodes of G belonging to some cycle are defined by:

$$\{x : \{U\} \mid \exists\, y : \{U\}(\mathit{IFP}(\phi(S), S)(x,y) \ \wedge \ x = y)\}.$$

Note that above, the existential quantification of y does *not* apply to variables occurring in $\phi(S)$, under the *IFP* operator.

Note that, in $\mathrm{CALC}_i^k + \mathit{IFP}$ and $\mathrm{CALC}_i^k + \mathit{PFP}$, there is no restriction on the arities of inductively defined relations. This is essential to the expressiveness results shown here.

The use of $\mathit{IFP}(\phi(S), S)$ as a term differs from the flat case (where such use is not possible). Although generally redundant, the use of such terms is essential in range-restricted versions of the fixpoint calculi, considered later.

It can be shown that many of the properties of the first-order logic extended with *IFP* and *PFP* continue to hold in the CALC_i^k extensions, including the collapse of the hierarchy based on the depth of nesting of the fixpoint operators.

The fixpoint calculi are closely related to deductive query languages, such as the Datalog extensions for complex objects considered in [AG91, Kup87, BNR+87]. The connection between fixpoint calculi and Datalog-like languages for complex objects is similar to that in the flat case (see [AV89]). For example, let inf-$\mathrm{Datalog}^{\neg}{}_i^k$ denote the straightforward extension to complex objects of $\mathrm{Datalog}^{\neg}$ with inflationary semantics, involving only variables of $< i,k >$-types. It can be shown that inf-$\mathrm{Datalog}^{\neg}{}_i^k$ has the same expressive power as $\mathrm{CALC}_i^k + \mathit{IFP}$. Thus, many of the expressiveness results proven here for the fixpoint extensions of CALC_i^k also apply to various deductive languages. The connection with deductive languages is studied in [GV91a].

In the next two sections we investigate the expressive power of the query languages for complex objects, described in this section. As in the classical relational case, we are aiming for characterizations of expressiveness in terms of complexity classes of queries.

4 Sparsity and Density

We begin by noting a discouraging fact: it is unlikely that one can provide meaningful characterizations of the expressive power of calculus-based queries in terms of complexity, without additional information. For instance, consider a CALC_i^k query. The variables of the query range over domains of $< i,k >$-types, so the query is evaluated in time polynomial in the size of such domains. However, the cardinality of the database may be very close to that of the domains, or it may be much smaller. If the database cardinality is close to the cardinality of the domains, then the query is evaluated in time polynomial in the size of the database. At the other extreme, if the database is very small compared to the domains, the query may take $P(hyper(i,k))$-time in the size of the database. The range of complexity of a given query with respect to different classes of inputs is so wide that no meaningful uniform characterization in terms of complexity is possible.

We consider additional information on the *inputs* which may limit the range of complexity of calculus-based queries. The information on the inputs concerns the way valid database inputs make use of $< i,k >$-types. This gives rise to the notions of "sparsity" and "density" of inputs with respect to a set of types. We claim that there are natural situations when such information on type usage by the database is available, and can be taken advantage of, when evaluating the complexity of queries. The expressiveness results in the presence of sparsity and density assumptions are presented next.

We begin by discussing the notions of "sparsity" and "density" of inputs with respect to given types. A class of inputs is sparse with respect to a type T if only few objects of type T are used in each input. As a consequence, the inputs are expected to be small relative to the cardinality of $dom(T)$. A class of inputs is dense with respect to a type T if it makes full use of the type T. Consequently, the cardinality of the database is expected to be close to that of $dom(T)$. Actually, we shall be interested in the looser notion of density and sparsity with respect to the *set* of $< i,k >$-types, rather than individual types. This is formalized as follows.

Definition 4.1 Let $\mathbf{R}$ be an $< i,k >$-database schema, and $\mathcal{I} \subseteq \mathbf{inst(R)}$. $\mathcal{I}$ is *dense* with respect to $< i,k >$-types if there exists a polynomial P such that, for each $\mathbf{I} \in \mathcal{I}$

$$|dom(i,k,atom(\mathbf{I}))| \leq P(|\mathbf{I}|).$$

$\mathcal{I}$ is *sparse* with respect to $< i,k >$-types if there exists a polynomial P such that, for each $\mathbf{I} \in \mathcal{I}$

$$|\mathbf{I}| \leq P(log(|dom(i,k,atom(\mathbf{I}))|)).$$

Density with respect to $< i,k >$-types requires the cardinality of the database to be within a polynomial of the cardinality of the domain of the $< i,k >$-types with respect to the constants in the database. Sparsity requires that the cardinality of the database be at most poly-logarithmic in the cardinality of the domain of $< i,k >$-types with respect to the constants in the database. Note that this implies that the cardinality of the database is at most polynomial in the domain of $< i-1,k >$-types. This indicates that the i-th level of nesting of the set operator is just cosmetic, and in fact can be avoided. Indeed, the objects of $< i,k >$-types can be represented using tuples (of fixed arity) of objects of $< i-1,k >$-type. Density and sparsity with respect to an individual type T can be defined similarly to the above. Then $|\mathbf{I}|$ is replaced in the definition by the cardinality of the set of (sub)-objects of type T in $\mathbf{I}$, and $|dom(i,k,atom(\mathbf{I}))|$ is replaced by $|dom(T,atom(\mathbf{I}))|$. Note that density and sparsity with respect to $< i,k >$-types do not imply density or sparsity with respect to a *specific* $< i,k >$-type.

It is often natural to make sparsity or density assumptions about databases. These are statements about the expected structure of the database. In some sense, they are similar to integrity constraints. In fact, they may be consequences of traditional constraints like functional dependencies. This is illustrated by the following.

Example 4.1 The VERSO model [AB86, Ver86] is a database model with nested relations, which requires relations at all levels of nesting to have an attribute of atomic type as a key. Thus, every VERSO schema defines a database which is sparse with respect to all higher types. More generally, sparsity with respect to a type T holds in the presence of functional dependencies ensuring that every component of type T is functionally determined by components of lower types.

Example 4.2 Consider a database which stores the sets of classes which may be taken by students. If there is no prerequisite structure, one can expect all combinations of classes to appear in the database. Thus, the set of valid database instances is dense with respect to the type set of classes. On the other hand, if there is a tight prerequisite structure (for instance, one that results in being able to take only a fixed maximum number of classes simultaneously), then the number of valid sets of courses which can be taken by students is polynomial in the number of classes. Thus, the set of valid database instances is sparse with respect to sets of classes.

For technical reasons, it is sometimes useful to consider notions of sparsity and density based on the *size* of $\mathbf{I}$ and $dom(i,k,atom(\mathbf{I}))$ rather than on their cardinalities, as in Definition 4.1. The definition of sparsity and density based on size is obtained by replacing everywhere cardinality by size. In the following, we use the terms *size-sparsity* and *cardinality-sparsity*, and *size-density* and *cardinality-density*, meaning sparsity and density with size and cardinality, respectively. We show next that the notions of density and sparsity based on size and cardinality are equivalent.

Lemma 4.1 Let $\mathbf{R}$ be an $< i,k >$-database schema and $\mathcal{I}$ a set of instances over $\mathbf{R}$. Then the following hold:

(i) $\mathcal{I}$ is cardinality-dense wrt $< i,k >$-types iff $\mathcal{I}$ is size-dense wrt $< i,k >$-types.

(ii) $\mathcal{I}$ is cardinality-sparse wrt $< i,k >$-types iff $\mathcal{I}$ is size-sparse wrt $< i,k >$-types.

Proof : Let $\mathbf{R}$ be an $< i,k >$-database schema, $\mathbf{I}$ an instance of $\mathbf{R}$, and $D = atom(\mathbf{I})$. The following easily checked facts are used:

(a) $|\mathbf{I}| \leq ||\mathbf{I}||$;

(b) $||\mathbf{I}|| \leq |\mathbf{I}|P(log(|dom(i,k,D)|))$, for some polynomial P;

(c) $||dom(i,k,D)|| \leq |dom(i,k,D)|P(log(|dom(i,k,D)|))$, by a straightforward extension of Proposition 2.1.

Consider (i). Let $\mathcal{I}$ be cardinality-dense wrt $< i,k >$-types. Then size-density follows from the following sequence of inequalities holding for each $\mathbf{I}$ in $\mathcal{I}$:

$$\begin{aligned}
||dom(i,k,D)|| &\leq |dom(i,k,D)|P(log(|dom(i,k,D)|)) \text{ for some fixed polynomial } P, \text{ by (c)}\\
&\leq Q(|\mathbf{I}|)P(log(Q(|\mathbf{I}|))) \text{ for some fixed polynomial } Q, \text{ by cardinality-density}\\
&\leq R(|\mathbf{I}|) \text{ for some fixed polynomial } R\\
&\leq R(||\mathbf{I}||), \text{ by (a).}
\end{aligned}$$

Conversely, suppose $\mathcal{I}$ is size-dense wrt $< i,k >$-types. We show that this implies cardinality-density. For each instance I in $\mathcal{I}$:

$$\begin{aligned}
|dom(i,k,D)| &\leq \|dom(i,k,D)\| \\
&\leq P(\|\mathbf{I}\|) \text{ for some polynomial P, by size-density} \\
&\leq P(|\mathbf{I}|Q(log(|dom(i,k,D)|)) \text{ for some polynomial Q, by (b)} \\
&\leq |\mathbf{I}|^m log^m(|dom(i,k,D)|) \text{ for some fixed } m.
\end{aligned}$$

Thus, we have

$$(\dagger) \quad |dom(i,k,D)| \leq |\mathbf{I}|^m log^m(|dom(i,k,D)|)$$

for some fixed m. We claim that, for all but finitely many I:

$$|dom(i,k,D)| \leq |\mathbf{I}|^{2m}.$$

To see this, note that:

$$\lim_{x\to\infty} \frac{x}{log^{2m}(x)} = \infty.$$

Thus, there exists M such that, for each $x > M$, $\frac{x}{log^{2m}(x)} \geq 1$, so $\frac{x^2}{log^{2m}(x)} \geq x$. Let $x = |dom(i,k,D)|$. Then for all but finitely many I,

$$\begin{aligned}
|dom(i,k,D)| &\leq \frac{|dom(i,k,D)|^2}{log^{2m}(|dom(i,k,D)|)} \\
&\leq |\mathbf{I}|^{2m} \text{ by } (\dagger).
\end{aligned}$$

Finally, one can clearly choose an appropriate j which covers also the finitely many instances for which the above does not hold, i.e.

$$|dom(i,k,atom(\mathbf{I}))| \leq |\mathbf{I}|^j$$

for all I in $\mathcal{I}$. This shows the cardinality-density of $\mathcal{I}$ wrt $< i,k >$-types, and concludes the proof of (i). Consider now (ii). Suppose $\mathcal{I}$ is cardinality-sparse wrt $< i,k >$-types. Then we have for each $\mathbf{I} \in \mathcal{I}$:

$$\begin{aligned}
\|\mathbf{I}\| &\leq |\mathbf{I}|P(log(|dom(i,k,D)|)) \text{ by (b)} \\
&\leq Q(log(|dom(i,k,D)|))P(log(|dom(i,k,D)|)) \text{ for some fixed polynomial } Q \\
&\qquad \text{by cardinality-sparsity} \\
&\leq R(log(|dom(i,k,D)|)) \text{ for some fixed polynomial } R \\
&\leq R(log(\|dom(i,k,D)\|)).
\end{aligned}$$

Conversely, suppose $\mathcal{I}$ is size-sparse. Then for each $\mathbf{I} \in \mathcal{I}$,

$$\begin{aligned}
|\mathbf{I}| &\leq \|\mathbf{I}\| \\
&\leq P(log(\|dom(i,k,D)\|)) \text{ for some fixed polynomial } P, \text{ by size-sparsity} \\
&\leq P(log(|dom(i,k,D)|\ R(log(|dom(i,k,D)|)))) \text{ for some fixed polynomial} R, \text{ by (c)} \\
&\leq Q(log(|dom(i,k,D)|) \text{ for some fixed polynomial } Q.
\end{aligned}$$

This concludes the proof. □

We next present our expressiveness results in the presence of density and sparsity assumptions. Let $\mathcal{I}$ be a set of instances over a database schema. If L is a query language, $L|\mathcal{I}$ denotes the set of queries expressed by L whose domains are restricted to $\mathcal{I}$. If $\mathcal{C}$ is a complexity class of queries, $\mathcal{C}|\mathcal{I}$ denotes the set of queries whose restrictions to $\mathcal{I}$ are in $\mathcal{C}$. We start with density. The main result of the section characterizes the expressive power of $\text{CALC}_i^k + \mathit{IFP}$ and $\text{CALC}_i^k + \mathit{PFP}$ in the presence of density assumptions on the database. The result that $\text{CALC}_i^k + \mathit{IFP}$ expresses precisely PTIME with this assumption is of special interest.

Theorem 4.1 Let $\mathbf{R}$ be an $< i,k >$-database schema $(i \geq 1, k \geq 2)$[3] and let $\mathcal{I}$, $\mathcal{I} \subseteq$ **inst**$(\mathbf{R})$ be dense with respect to $< i,k >$-types. Then :

1. $\text{CALC}_i^k|_\mathcal{I} \subset \text{P}(log)\text{-space}|_\mathcal{I}$[4],
2. $\text{CALC}_i^k + \mathit{IFP}|_\mathcal{I} = \text{PTIME}|_\mathcal{I}$,
3. $\text{CALC}_i^k + \mathit{PFP}|_\mathcal{I} = \text{PSPACE}|_\mathcal{I}$.

We present a sequence of lemmas which yields the proof. The proof has two aspects. The complexity bounds (i.e. the inclusion of the three classes of queries in the respective complexity classes) are obtained straightforwardly using the density assumption. For (2) and (3), the converse inclusions involve showing expressibility of the queries in $\text{PTIME}|_\mathcal{I}$ and $\text{PSPACE}|_\mathcal{I}$ by $\text{CALC}_i^k + \mathit{IFP}$ and $\text{CALC}_i^k + \mathit{PFP}$, respectively. We begin by showing the complexity bounds.

Lemma 4.2 With the assumptions in Theorem 4.1,

1. $\text{CALC}_i^k|_\mathcal{I} \subset P(log)\text{-space}|_\mathcal{I}$,
2. $\text{CALC}_i^k + \mathit{IFP}|_\mathcal{I} \subseteq \text{PTIME}|_\mathcal{I}$,
3. $\text{CALC}_i^k + \mathit{PFP}|_\mathcal{I} \subseteq \text{PSPACE}|_\mathcal{I}$.

Proof: Consider the inclusion $\text{CALC}_i^k|_\mathcal{I} \subset \text{P}(log)\text{-space}|_\mathcal{I}$. Let ϕ be a CALC_i^k formula over $\mathbf{R}$, and $\mathbf{I}$ an instance of $\mathbf{R}$. By the semantics of CALC_i^k formulas, variables in ϕ range over $dom(i,k,atom(\mathbf{I}))$. It follows that the recognition problem associated with ϕ is in space polylogarithmic in the size of the domains of $< i,k >$-types. This is shown in the same way as the inclusion $\text{CALC}_0^0 \subset P(log)\text{-space}$ for the flat relational calculus (see [Var82]). The density of $\mathcal{I}$ with respect to $< i,k >$-types then states that $\|dom(i,k,atom(\mathbf{I}))\| \leq P(\|\mathbf{I}\|)$ for some fixed polynomial P. It follows that ϕ is also in polylogarithmic space with respect to the size of inputs $\mathbf{I}$ in $\mathcal{I}$. The proofs of (2.) and (3.) are similar extensions of the results known for the flat case that *fixpoint* $\subseteq$ PTIME and *while* $\subseteq$ PSPACE [Var82]. $\square$

We now turn to the second part of the proof, the converse inclusions in (2.) and (3.). We have to show that all queries in $\text{PTIME}|_\mathcal{I}$ and $\text{PSPACE}|_\mathcal{I}$ are expressible by $\text{CALC}_i^k + \mathit{IFP}$

[3] Note that the statement still holds for $i = 2$ and $k = 1$. It is also the case of the other results of the paper.

[4] This refers to the recognition problem.

and $\text{CALC}_i^k + PFP$, respectively. The proofs are essentially extensions to the complex object case of the simulations, in the flat case, of the PTIME and PSPACE queries by *fixpoint* and *while*, in the presence of an order on the domain [Var82, Imm86]. In both cases, the core of the proof involves the simulation of TM computations. The order assumed in the flat case is essential to represent the tape of the simulated TM. The main difference in the complex object case is that no order assumption is needed, since an order can be postulated within the languages. We will only provide the proof for (2.), since the same techniques are used for (3.).

A key aspect of the proof consists of showing that one can define orderings of the domains of $< i, k >$-types using CALC_i^k formulas. The orderings are constructed inductively, starting from an ordering $<_U$ on the atomic constants. The formulas defining the orderings, as well as the other formulas used in the proof, depend on $<_U$. Eventually, the entire simulation is expressed by some formula $\psi(<_U)$ depending on $<_U$. The ordering $<_U$ is provided by the formula:

$$\exists <_U : \{U, U\}[order(<_U) \wedge \psi(<_U)]$$

where $order(<_U)$ is the following CALC_1^2 formula stating that $<_U$ is an order on $dom(U)$

$$\forall x : U \forall y : U \forall z : U[x <_U x \wedge (x <_U y \vee y <_U x) \wedge ((x <_U y \wedge y <_U z) \rightarrow x <_U z)$$

$$\wedge((x <_U y \wedge y <_U x) \rightarrow x = y)].$$

Note that in the final formula, $<_U$ is a variable of type $< 1, 2 >$. This explains the requirement that $i \geq 1$ and $k \geq 2$ in the statement of Theorem 4.1.

The following makes more precise the notion of ordering induced on the domain of a type T by an ordering on the atomic constants.

Definition 4.2 Let D be a finite set of constants and $<_U$ a binary relation holding a total order on $dom(U, D)$. For each type T, the *order* on $dom(T, D)$ *induced* by $<_U$, denoted $<_T$, is defined inductively as follows[5]:

- if $T = [T_1, ..., T_k]$, then for each $o_1, o_2 \in dom(T, D)$, $o_1 <_T o_2$ iff there exists i, $1 \leq i \leq k$, such that $o_1.j = o_2.j$ for all $j < i$ and $o_1.i <_{T_i} o_2.i$.
- if $T = \{S\}$, then for each $o_1, o_2 \in dom(T, D)$, $o_1 <_T o_2$ iff

  $$max_{<_S}(o_1 - o_2) <_S max_{<_S}(o_2 - o_1),$$

  where $max_{<_S}(o_i - o_j)$ denotes the maximum element (of type S) belonging to $o_i - o_j$, wrt the order $<_S$ on objects of type S.

We next show that one can define in CALC_i^k the order relations $<_T$ for each $< i, k >$-type T, starting from an order $<_U$ on the atomic constants.

Lemma 4.3 Let D be a finite set of constants. Then, for every $< i, k >$-type T ($i \geq 1, k \geq 2$), there exists a CALC_i^k formula $\phi_{<_T}$, which, given $<_U$, defines the order relation $<_T$ on $dom(T, D)$.

[5] $x <_T y$ stands for $<_T (x, y)$

Proof : The proof is by induction on the $< i, k >$-type T. The induction follows Definition 4.2 for $<_T$. For $T = U$, we assume the order $<_U$ on atomic constants is given, and $\phi_U(x, y) = x <_U y$. Next, suppose $T = [T_1, ..., T_k]$. By the induction hypothesis, there are CALC$_i^k$ formulas $\phi_{<_{T_i}}$ defining $<_{T_i}$ given $<_U$. Then:

$$\phi_{<_T}(x, y) = \bigvee_{1 \leq i \leq k} (\bigwedge_{j<i} (x_j = y_j) \wedge \phi_{<_{T_i}}(x_i, y_i)).$$

If $T = \{S\}$ then:

$$\phi_{<_T}(x, y) = \exists z : S \exists z' : S \ (Max_{<_S}(x - y, z) \ \wedge \ Max_{<_S}(y - x, z') \ \wedge \ z <_S z'),$$

where $Max_{<_S}(x - y, z)$ is a CALC$_i^k$ formula stating that z is the maximum element in $x - y$ wrt $<_S$. □

The proof of (2.) involves simulating a Turing Machine on databases encoded on its tape. Recall that a database instance is presented as input to a Turing Machine in a standard encoding, where each atomic constant is represented in binary notation. This is necessary because the tape alphabet is finite. To simulate TM computations we will need to construct the encodings of objects of $< i, k >$-types. Therefore, it is useful to first construct, for each $< i, k >$-type T, a CALC$_k^i$ + *IFP* formula which defines the standard encoding of objects of type T with respect to some finite set D of atomic constants. Note that each atomic constant requires $log(n)$ bits, where $n = |D|$. Next, for each object o of an $< i, k >$-type T $(i \geq 1)$,

$$||o|| \ \leq \ log^m(|dom(i, k, D)|) \ = \ |dom(i - 1, k, D)|^m \text{ for some } m.$$

Thus, we need to represent a "dictionary" holding a word of length $|dom(i - 1, k, D)|^m$ for each object of type T. We can use an m-tuple of objects in $dom(i - 1, k, D)$ to represent the entries for each object of type T. Thus, we will use a $m + 2$-ary relation $CODE_T$ where a tuple $[o, \vec{i}, x]$ indicates that x is the $\vec{i}$-th symbol in the word $enc(o)$. Here o is an object of type T, $\vec{i}$ is an m-tuple of objects in $dom(i - 1, k, D)$, and x is a tape symbol of the TM. The order on the m-tuples $\vec{i}$ is that induced by some fixed ordering $<_U$ on atomic constants. The following lemma shows that such encodings can be defined in CALC$_i^k$+*IFP*.

Lemma 4.4 Let **R** be an $< i, k >$-database schema as in the statement of Theorem 4.1. Let D be a finite set of atomic constants. For every $< i, k >$-type T $(i \geq 1, k \geq 2)$, there exists a CALC$_i^k$+*IFP* formula over $\mathbf{R} \cup <_U$ defining the relation $CODE_T$ wrt the ordering $<_U$.

Proof : Let i and k be fixed, $i \geq 1, k \geq 2$. Let D be a finite set of atomic constants and $<_T$ the orderings induced on $dom(i,k,D)$ by $<_U$. Consider first the construction for $T = U$. $CODE_U$ is a ternary relation. We provide a formula with three free variables defining $CODE_U$. Suppose $|D| = n$. The binary encoding of n constants requires $\log n$ bits for each constant. So, we can use the n (ordered) constants themselves to identify the digits of the binary encoding of each constant. Thus, the meaning of $CODE_U(o,i,x)$ is that in the encoding of constant o, the i's digit is x. The figure shows the relation $CODE_U$ for five constants $\{a,b,c,d,e\}$ (wrt the order $abcde$). Note that this relation is not explicitly named. Instead, the formula defining it is used whenever needed.

CODE		
constant	index	digit
a	a	0
b	a	1
c	a	1
c	b	0
d	a	1
d	b	1
e	a	1
e	b	0
e	c	0

The relation $CODE_U$ can be defined in CALC$_i^k$+*IFP* as follows. First, the relation contains the tuple $[a,a,0]$. By induction, consider the largest value α of the first argument of $CODE_U$. We add the tuples whose first argument is the successor of α, say β, and whose second and third argument are defined as follows. Consider the largest γ such that $CODE(\alpha,\gamma,0)$. Either it exists, and we add the values $[\beta,y,d]$ for each $CODE(\alpha,y,d)$ where $y < \gamma$ and the value $[\beta,\gamma,1]$ and the values $[\beta,y,0]$ for each $CODE(\alpha,y,1)$ and each $y > \gamma$. If there is no such γ, then we add the following tuples : $[\beta,a,1]$ and $[\beta,y,0]$ for each $b \leq y \leq \delta$, where $\delta = succ(max\{z/\exists t\ CODE[\alpha,z,t]\})$ ($succ$ and max are defined wrt $<_U$). This can obviously be expressed in CALC$_i^k$+*IFP*.

The above construction can be extended to formulas defining $CODE_T$ for objects of higher types. We proceed recursively as follows. Suppose we already defined a relation $CODE_T$ for some type T. We now define $CODE_{\{T\}}$. For each set $a = \{a_1,...,a_n\} : \{T\}$, we add the following tuples to the relation. First, the tuple $[a,\alpha,\{]$ is added, where α is the smallest element of type T. Then we add the encoding of the smallest element of a, say a_1, by increasing the indices. We then add the "#" with the next index, and continue with a_2, etc. When the set is exhausted we add the symbol "}". The construction is similar for tuple types. □

We now complete the proof of Theorem 4.1.

Proof of Theorem 4.1 (2.) : Let **R** be an $< i,k >$-database schema ($i \geq 1, k \geq 2$) and $\mathcal{I}$ a family of instances of **R** which is dense wrt $< i,k >$-types. We elaborate on the inclusion PTIME$|_\mathcal{I} \subseteq$ CALC$_i^k$+*IFP*$|_\mathcal{I}$ of (2).

We have to show that CALC$_i^k$ + *IFP* can simulate each query in PTIME over inputs in $\mathcal{I}$. Let q be such a query. By definition, there exists a Turing Machine M which, on input enc(**I**) terminates in polynomial time with output $enc(q(\mathbf{I}))$. Thus, we will construct a CALC$_i^k$+*IFP* formula $\phi(<_U)$ which, on input $\mathbf{I} \in \mathcal{I}$ does the following:

- simulate the computation of M on input enc(**I**), and
- decode the result q(**I**) from its encoding $enc(q(\mathbf{I}))$.

We next focus on the simulation of M. We need to represent a configuration of M as a relation. In particular, the tape has to be represented. Since the tape is infinite, we only represent the finite portion, polynomial in length, which is potentially used. We need a way to identify each cell of the tape. Notice that there exists some h such that M on input $enc(\mathbf{I})$ takes time $||\mathbf{I}||^h$, so uses in its computation at most $||\mathbf{I}||^h$ tape cells. Let $D = atom(\mathbf{I})$. Recall that $||\mathbf{I}|| \leq |dom(i,k,D)|^j$ for some j. Thus, each cell can be uniquely identified by an m-tuple of constants from $dom(i,k,D)$, for some m. The ordered m-tuples allow to represent a sequence of cells, hence the tape. Furthermore, to represent a configuration at time t, another m-tuple can be used as a timestamp. Note that one cannot remove the tuples representing old configurations of M, due to the inflationary nature of $\mathrm{CALC}_i^k + IFP$ computations. Thus, to keep track of the sequence of configurations in a computation of M, one can use a $(2m+2)$-ary relation R_M where the first m columns serve as a timestamp for the configuration, the next m identify the tape cells, the $(2m+1)$-st column holds the content of the cell, and the $(2m+2)$-nd indicates the state and position of the head. Note that m may be larger than k. However, we never refer explicitly to R_M, but only use the formula defining it, whenever needed. The fact that the *IFP* operator is not restricted by k and can thus define relations of arbitrary arity, is essential here.

We are dealing now with a "double" encoding: the database is encoded on the tape, then the tape is encoded back into R_M. To illustrate this, consider again the instance I of Figure 1 and its encoding $enc(\mathbf{I})$ on the tape of M in Figure 2. Notice that this involves a $< 1,2 >$-database schema. The initial configuration with the same tape content, where the state is s and the head points to the first cell of the tape, is represented in the relation shown in the figure below. The representation assumes that $m = 4$, so the arity of the relation is 10. For simplicity, we denote by $\vec{i}_j$ the j-th m-tuple of objects in $dom(0,2,\{a,b,c\})$ in the order induced by the enumeration abc. For instance, $\vec{i}_1 = [a,a,a,a]$ and $\vec{i}_6 = [a,a,b,c]$. Only the first few tuples of the representation are shown.

For a given instance I, the simulation of M proceeds in two phases:

(†) compute, in R_M, a representation of the initial configuration of M on input $enc(\mathbf{I})$;

(‡) compute the sequence of consecutive configurations of M until termination, using the representation provided by R_M.

The construction of $enc(\mathbf{I})$ in (†) is essentially the same as that for $CODE_T$ in Lemma 4.4. We next outline the construction for (‡). One has to simulate the computation of M starting from the initial configuration represented in R_M. To construct a new configuration from the current one, one has to simulate a move of M. This is repeated until M reaches a final state (accepting or rejecting), which, as we assumed earlier, happens after at most $|dom(i,k,atom(\mathbf{I}))|^m$ steps. The iteration can be performed using the fixpoint operator in CALC_i^k+*IFP*. Each step consists of defining the new configuration from the current one, timestamping it, and adding it to R_M. This can be done with a CALC_i^k+*IFP* formula.

For instance, suppose the current state of M is q, the content of the current cell is 0, and the corresponding move of M is to change 0 to 1, move right, and change states from q to r. Suppose $\vec{t}$ is the timestamp identifying the current configuration, and R_M contains the tuple $[\vec{t}, \vec{j}, 0, q]$. The tuples describing the new configuration of M are the tuples $[\vec{t'}, \vec{i}, x, y]$ such that $\vec{t'}$ is the successor of $\vec{t}$ and

(a) $\vec{i}$ is less than $\vec{j}$ or more than the successor of $\vec{j}$, and $[\vec{t}, \vec{i}, x, y] \in R_M$, or

(b) $\vec{i} = \vec{j}$, $x = 1$, and $y = 0$, or

(c) $\vec{i}$ is the successor of $\vec{j}$ and $y = r$ and $[\vec{t}, \vec{i}, x, 0] \in R_M$.

In other words, (a) says that the cells other than the j-th cell and its successor remain unchanged; (b) says that the content of cell j changes from 0 to 1, and the head no longer points to the j-th cell; finally, (c) says that the head points to the successor of the jth cell, the new state is r, and the content x of the cell is unchanged. Clearly, (a)-(c) can be expressed by a CALC_i^k+*IFP* formula. One such formula is needed for each instruction of M, and the formula corresponding to the finite set of instructions is obtained by their disjunction.

R_M			
$\vec{i}_1$	$\vec{i}_1$	P	s
$\vec{i}_1$	$\vec{i}_2$	[	0
$\vec{i}_1$	$\vec{i}_3$	0	0
$\vec{i}_1$	$\vec{i}_4$	1	0
$\vec{i}_1$	$\vec{i}_5$	#	0
$\vec{i}_1$	$\vec{i}_6$	{	0
$\vec{i}_1$	$\vec{i}_7$	0	0
$\vec{i}_1$	$\vec{i}_8$	0	0
$\vec{i}_1$	$\vec{i}_9$	#	0
$\vec{i}_1$	$\vec{i}_{10}$	0	0
$\vec{i}_1$	$\vec{i}_{11}$	1	0
$\vec{i}_1$	$\vec{i}_{12}$	}	0
.	.	.	.
.	.	.	.
.	.	.	.

Representation of a configuration of M at time $\vec{i}_1$

With (†) and (‡) achieved, it remains to decode the representation of $enc(q(\mathrm{I}))$ in R_M to obtain the result. This is essentially the inverse of the encoding process. It can be easily verified that this can be achieved using a CALC_i^k+*IFP* formula.

Note that the density assumption is not needed for the simulation of queries in PTIME, but only to guarantee inclusion in PTIME (see proof of Lemma 4.2).

The proof that $\mathrm{PSPACE}|_{\mathcal{I}} \subseteq \mathrm{CALC}_i^k + PFP|_{\mathcal{I}}$ is done similarly. The difference lies in the fact that the CALC_i^k+*PFP* computation needs not be inflationary, unlike CALC_i^k+*IFP* computations. This simplifies the simulation. For instance, only the tuples corresponding to the *current* configuration of M are kept in R_M, so no timestamping is required. □

The characterization of the PTIME queries over dense inputs deserves some discussion. No language expressing precisely the PTIME queries is known. Previous characterizations of the PTIME queries over flat relations involve trade-offs such as access to an order [Imm86, Var82], or non-determinism [AV91, ASV90]. In a dense, non-flat database, the order assumption is no longer necessary, because the inputs are large enough to allow the efficient construction of the necessary orderings of the domains of types, induced by orderings of the atomic constants. In this respect, the result can be viewed as extending the result of [Imm86, Var82] for the *fixpoint* queries with order. However, the density assumption itself is specific to complex objects and there is no analog for flat databases.

We now present an extension of Theorem 4.1 involving both sparsity and density assumptions.

Theorem 4.2 Let **R** be an $< i,k >$-database schema ($i \geq 1, k \geq 2$) and $\mathcal{I}$, $\mathcal{I} \subseteq$ **inst(R)** be dense with respect to $< i-j,k >$-types, and sparse with respect to $< i-j+1,k >$-types, for some j, $1 \leq j \leq i$. Then :

1. $\text{CALC}_i^k|_{\mathcal{I}} \subset \text{P}(hyper(j-1,k))\text{-space}|_{\mathcal{I}}$,
2. $\text{CALC}_i^k + IFP|_{\mathcal{I}} = \text{P}(hyper(j,k))\text{-time}|_{\mathcal{I}}$,
3. $\text{CALC}_i^k + PFP|_{\mathcal{I}} = \text{P}(hyper(j,k))\text{-space}|_{\mathcal{I}}$.

Proof : This is a straightforward extension of the previous result. Consider a class of inputs $\mathcal{I}$, dense with respect to $< i-j,k >$-types, and sparse with respect to $< i-j+1,k >$-types, for some j, $1 \leq j \leq i$.

Consider (1.). CALC_i^k is in space polylogarithmic in the size of the domains of $< i,k >$-types. Since $\mathcal{I}$ is dense with respect to $< i-j,k >$-types, the size of the domains of $< i,k >$-types is polynomially equivalent to $hyper(j,k)(||\mathbf{I}||)$, for each $\mathbf{I} \in \mathcal{I}$. Therefore, $\text{CALC}_k^i|_{\mathcal{I}}$ is in space polynomial in $hyper(j-1,k)$ in the size of the input.

Next, consider (2.). The inclusion $\text{CALC}_i^k + IFP|_{\mathcal{I}} \subseteq \text{P}(hyper(j,k))\text{-time}|_{\mathcal{I}}$ is in the same spirit as above. For the reverse inclusion $\supseteq$, the construction is similar to that in (2.) of Theorem 4.1. Note that the number of steps and cells required by a query in $\text{P}(hyper(j,k))$-$\text{time}|_{\mathcal{I}}$ is bounded by a polynomial in $hyper(j,k)(||\mathbf{I}||)$ for each $\mathbf{I} \in \mathcal{I}$, i.e. is bounded by $hyper(i,k)(|D|)$. Therefore, the machine can be simulated by a query in $\text{CALC}_i^k + IFP$, since $hyper(i,k)(|D|)$ indices for the tape cells and timestamps can be defined, similarly to the proof of Theorem 4.1.

The proof of (3.) is similar. □

Remark 4.1 In the above, sparsity and density were considered with respect to domains of types built using a single sort of constants. In practice, density and sparsity are more likely to hold relative to types over particular sorts, rather than the entire set of constants. For instance, a database involving *employees*, *days-of-the-week*, and *departments*, might be sparse with respect to sets of *employees* but dense with respect to sets of *days-of-the-week* and sets of *departments*. The complexity results above then indicate that queries may use variables of type set of *days-of-the-week* and set of *departments* without a prohibitive cost in complexity (relative to the size of the database), but that quantifying over sets of *employees* is not recommended.

Sparsity and density were used above to obtain expressiveness results relating query languages to complexity classes. It turns out that these assumptions are also relevant to the *relative* expressive power of query languages. Such a result, showing the equivalence of two range-restricted languages in the presence of sparsity assumptions, is provided at the end of the next section (Proposition 5.2).

5 Safe and Range-Restricted Languages

In the previous section we examined the impact on query complexity of type usage by the database, captured by the notions of sparsity and density. We next present another factor which may limit the range of complexity of queries. It is based on restrictions on the queries rather than on information about the database.

The active domain semantics ensures the finiteness of the answer to a query in any of the languages considered so far. Nevertheless, due to the complex objects, computing the answer may require high time/space complexity with respect to the size of the input, typically hyperexponential. In this section we restrict the complex object languages to bound the computation of the queries to a given time/space complexity in the size of the input. We first define a semantic concept, *safety* with respect to a complexity class C, by restricting the range of the variables to a domain computable from the input database with complexity C in the size of the input. We then show that we can enforce safety with respect to PTIME and PSPACE by a syntactic notion of range restriction.

Recall that in the active domain semantics, a variable of type T ranges over the set $dom(T, D)$, where D is the set of atomic constants appearing either in the database or in the query itself. If q is a query over a schema $\mathbf{R}$ and $\mathbf{I}$ a database instance over $\mathbf{R}$, the active domain interpretation of q on $\mathbf{I}$ is denoted by $q(\mathbf{I})_{ad}$.

The *restricted domain semantics* is based on domains of interpretation of the variables defined by a *range function*. Let q be a query over an input schema $\mathbf{R}$. A *range function* r_q for q associates to each variable[6] x of type T in q and each instance $\mathbf{I}$ over $\mathbf{R}$, a set of objects of type T such that $r_q(x)(\mathbf{I}) \subseteq dom(T, D)$, where D is the set of constants appearing either in q or in $\mathbf{I}$. When q is applied to $\mathbf{I}$, variable x ranges over $r_q(x)(\mathbf{I})$. The restricted domain interpretation of q on $\mathbf{I}$ with range function r_q is denoted by $q(\mathbf{I})_{r_q}$.

Note that the active domain semantics is a special case of the restricted domain semantics with a range function associating $dom(T, D)$ to all variables of type T. Clearly, the restricted domain semantics allows more flexibility than the active domain semantics.

Consider a query q on an input schema $\mathbf{R}$ in a language L and let C be a time/space complexity class. A range function r_q is in C if for each variable x in q, $r_q(x)$ is a function on the set of instances over $\mathbf{R}$ computable in C. We can now define *C-safe queries*.

Definition 5.1 A query q over $\mathbf{R}$ is *C-safe* if there exists a range function r_q in C such that for each instance $\mathbf{I}$ over $\mathbf{R}$, the active domain interpretation is equivalent to the restricted domain interpretation with range function r_q, i.e.

$$q(\mathbf{I})_{r_q} = q(\mathbf{I})_{ad}.$$

For each language L and each complexity class C, we denote by SAFE$_C$-L the set of C-safe queries in L. In many cases, safety of a query wrt a complexity class C limits its complexity to C. Thus we have the following fact, whose proof is immediate.

Proposition 5.1 :

1. SAFE$_{LOGSPACE}$-CALC $\subseteq$ LOGSPACE,

[6]Without loss of generality, we assume in the following that no variable symbol occurs both free and bound, or is bound by more than one quantifier.

2. SAFE_{PTIME}-(CALC+*IFP*) $\subseteq$ PTIME,

3. SAFE_{PSPACE}-(CALC+*PFP*) $\subseteq$ PSPACE.

It is easy to see that, in general, $\text{SAFE}_{\mathcal{C}}$-$L$ is not recursive. Thus, there is no checkable syntactic restriction that completely characterizes safe queries wrt a complexity class in a given language. We next consider a syntactic criterion called *range restriction*, defined in [GV91b], and show that it provides *sufficient* conditions for safety wrt PTIME and PSPACE, for CALC and its fixpoint extensions. We shall see that, under certain assumptions, range-restricted queries express *all* safe queries wrt PTIME and PSPACE.

We first define inductively range-restricted formulas in CALC without the fixpoint operator, then extend the definition to CALC + *IFP* and CALC + *PFP*. First, we define range-restricted variables in CALC formulas (by abuse of terminology, variables include the projections $x.i$ of tuple variables x on the i-th component). The set of variables x or $x.i$ in a formula ϕ is denoted $var(\phi)$.

Definition 5.2 Let ϕ be a formula in CALC. The set of *range restricted variables* in ϕ, denoted $RR(\phi)$, is the set of variables occurring in ϕ whose range restriction in ϕ can be inferred using the following rules applied to subformulas of ϕ:

1. if ξ is an atomic formula $R(x_1, ..., x_n)$, where R is a database relation, then $RR(\xi) = var(\xi)$;

2. if $x \in RR(\xi)$ and x is of type $[T_1, ..., T_m]$, then for each i such that $1 \leq i \leq m$, $x.i \in RR(\xi)$;

3. if $x \in var(\xi)$, x is of type $[T_1, ..., T_m]$, and $x.i \in RR(\xi)$ for each i such that $1 \leq i \leq m$, then $x \in RR(\xi)$;

4. if ξ is of the form $(x = y) \wedge \phi'$ or $(x \in y) \wedge \phi'$, and $y \in RR(\phi')$, or of the form $x = c$ where c is a constant, then $x \in RR(\xi)$;

5. if $\xi = (\phi_1 \wedge \phi_2)$ and $x \in RR(\phi_1)$ or $x \in RR(\phi_2)$, then $x \in RR(\xi)$;

6. if $\xi = (\phi_1 \vee \phi_2)$ and $x \in var(\phi_i)$ implies $x \in RR(\phi_i)$ for $i \in \{1, 2\}$, then $x \in RR(\xi)$;

7. if $\xi = \forall x : T(\phi')$ and $x \in RR(\neg\phi')$[7], then $x \in RR(\xi)$;

8. if $\xi = \exists x : T(\phi')$ and $x \in RR(\phi')$, then $x \in RR(\xi)$;

9. if $\xi = \forall y : T(y \in x \Leftrightarrow \phi'(y))$, $y \in RR(\phi')$, then $x \in RR(\xi)$.

A formula in CALC is *range restricted* if all its variables are range restricted. The set of range-restricted formulas in CALC is denoted *RR*-CALC. Range-restricted formulas in CALC correspond to strictly safe formulas in [AB87].

[7] The expression $\neg\phi$ denotes the formula where the negations are pushed past all quantifiers and connectives.

Example 5.1 The following formula defines the *nest* of the second argument of a binary relation P:

$$\{(x : U, s : \{U\})/\exists z : U(P(x,z)) \wedge \forall y : U(P(x,y) \Leftrightarrow y \in s)\}.$$

It is easily shown that this formula is range restricted.

Condition 9 in Definition 5.2 for CALC is ad hoc and lacks elegance. It is needed to compute limited grouping, like the nest in the above example. We will show that this condition can be discarded by using the fixpoint operator in a range-restricted fashion. The range-restricted calculus extended with the fixpoint operator is defined next. The definition is slightly complicated due to the following facts:

- in an inductively defined relation $IFP(\phi(R), R)(z_1, ..., z_n)$, some of the columns may be range restricted while others may not, and
- subformulas occurring within the scope of IFP operators contain, in addition to database relations, the relations bound to IFP operators.

We therefore must define the range-restricted arguments, or columns, of a relation defined inductively by a fixpoint operator. To deal with the second point formally, the following terminology will be useful.

Notation: The fixpoint operator IFP can be viewed as a form of quantification over relations. In this spirit, we say that the *scope* of IFP in a formula $IFP(\phi(S), S)$ is ϕ. Also, S is said to be *bound* to this occurrence of IFP. A relation is *free* in a formula if it is not a database relation and is not bound to any IFP operator. Note that there are no free relations in a CALC+IFP formula. However, subformulas of CALC+IFP formulas may have free relations. □

The definitions of range-restricted columns and range-restricted variables in a formula are mutually recursive. We define the range-restricted variables for subformulas with free relations, relative to given range-restricted columns for them. This yields as a special case the definition for CALC+IFP formulas with no free relations.

Definition 5.3 Let ϕ be a subformula of a CALC+IFP formula over database schema **R**, with free relations **S**. Let τ be a mapping providing, for each $S \in \mathbf{S}$, a subset of its columns (the range-restricted ones). A variable is range restricted in ϕ relative to τ (denoted $RR_\tau(\phi)$) if this can be inferred using the following rules applied to the subformulas of ϕ :

1-8: as in Definition 5.2, where "$RR(\xi)$" is replaced by "$RR_\tau(\xi)$" and point (1.) is augmented by:

1'. if ξ is an atomic formula $S(z_1, ..., z_n)$, where $S \in \mathbf{S}$ and x occurs in some column of S in $\tau(S)$, then $x \in RR_\tau(\xi)$.

It remains to deal with the following cases:

9'. if ξ is $x = IFP(\phi(R), R)$, then:
$RR_\tau(IFP(\phi(R), R)(z_1, ..., z_n)) \subseteq RR_\tau(\xi)$, where the z_i are the free variables of ϕ;
if all columns of $IFP(\phi(R), R)$ are range restricted relative to τ, then $x \in RR_\tau(\xi)$.

10. if $\xi = IFP(\phi(S), S)(z_1, ..., z_n)$, we define the set of range-restricted columns for S at each point in the inductive definition of S, as follows. We denote by τ_i the extension of τ to S at the i-th step in the inductive definition of S. More precisely,

$$\begin{array}{rcl} \tau_0(S) & = & \{1, ..., n\}, \\ \tau_i(S) & = & \tau_{i-1}(S) - \{i \mid z_i \notin RR_{\tau_{i-1}}(\phi)\} \text{ for } i > 0. \end{array}$$

Note that the sequence $\{\tau_i(S)\}_i$ is non-increasing. Since S has finitely many columns, there exists $k \geq 0$ such that $\tau_k(S) = \tau_{k+1}(S)$. Let us denote τ_k by τ^*. Then the set of range-restricted columns of ξ (relative to τ) consists of those columns in $\tau^*(S)$ and

$$RR_{\tau^*}(\phi) \subseteq RR_\tau(\xi).$$

The following illustrates the definition.

Example 5.2 Let P be a unary relation and S a ternary relation, where P is a database relation, and S is defined inductively using an *IFP* operator. Consider the formula $\xi = IFP(\phi(S), S)(x, y, z)$ where:

$$\phi(S)(x,y,z) = \exists t : U(S(z,x,t) \wedge S(t,y,y)) \vee (\neg P(x) \wedge P(y))$$

By definition, $RR(\xi) = RR_{\tau^*}(\phi)$, where $\tau^*(S)$ provides the range-restricted columns of $IFP(\phi(S), S)(x, y, z)$, defined inductively as follows.

$$\begin{array}{rclrcl} \tau_0(S) & = & \{1,2,3\}, & RR_{\tau_0}(\phi) & = & \{y,z,t\}, \\ \tau_1(S) & = & \{2,3\}, & RR_{\tau_1}(\phi) & = & \{y,t\}, \\ \tau_2(S) & = & \{2\}, & RR_{\tau_2}(\phi) & = & \{y\}, \\ \tau_3(S) & = & \tau_2(S), & RR_{\tau_3}(\phi) & = & RR_{\tau_2}(\phi). \end{array}$$

Thus, $\tau^*(S) = \tau_2(S) = \{2\}$ and $RR(\xi) = \{y\}$.

A CALC+*IFP* formula is *range restricted* iff all its variables are range restricted. The set of range-restricted CALC+*IFP* formulas is denoted by *RR*-(CALC+*IFP*). Note that *RR*-(CALC+*PFP*) is defined in the same way as *RR*-(CALC+*IFP*) with *PFP* instead of *IFP*.

The next example shows how the nest of a relation can be defined by a range-restricted formula of CALC+*IFP*.

Example 5.3 Let P be a binary relation. The following formula defines the *nest* on the second argument, using a unary relation Q and variables $x : U$ and $s : \{U\}$.

$$\{(x : U, s : \{U\})/\exists z : U(P(x,z)) \wedge s = IFP((P(x,y) \vee Q(y)), Q)\}.$$

Note how the fixpoint operator is used as a relational term. The fixpoint is reached here in one step.

We next show that range restriction of the calculus and its fixpoint extensions guarantees C-safety if C is LOGSPACE, PTIME or PSPACE.

Theorem 5.1 :

(a) RR-CALC $\subseteq$ SAFE$_{LOGSPACE}$-CALC,

(b) RR-(CALC+*IFP*) $\subseteq$ SAFE$_{PTIME}$-(CALC+*IFP*),

(c) RR-(CALC+*PFP*) $\subseteq$ SAFE$_{PSPACE}$-(CALC+*PFP*).

Proof : The proof for each of (a), (b) and (c) consists of three parts:

(i) for each formula ϕ in RR-CALC, RR-(CALC+*IFP*) or RR-(CALC+*PFP*), define a corresponding range function $r_\phi(x)$, for each variable x in ϕ;

(ii) show the equivalence of $(\phi)_{ad}$ and $(\phi)_{r_\phi}$;

(iii) show that each range function $r_\phi(x)$ is in LOGSPACE, PTIME or PSPACE, respectively.

We focus on (a) and (b), since (c) is similar to (b). The definition of the range functions is based on a proof of range-restriction of the corresponding variable. The proof uses the rules in the definition of range restriction. Note that, just as there may be several ways to prove range-restriction of a given variable, there are several range functions which could be defined for it, corresponding to the different proofs. To select one particular range function for a variable, it is sufficient to select arbitrarily one proof of its range-restriction. This can be done for instance by ordering the proofs based on the rules used, and choosing the minimum one. We are not concerned with the particular choice here, since any will do. Instead, we provide the range function corresponding to any proof of range-restriction for each variable.

Consider RR-CALC. Let $\xi \in$ RR-CALC and x be a variable in ξ. Consider a proof of the range-restriction of x, denoted by a sequence $i_1 ... i_n$ of rules (from Definition 5.2) used in the proof. The range function is defined by induction on n. We also prove by induction (ii) and (iii). The base case ($n = 1$) corresponds to rule 1, since the only proof of length 1 consist of applying rule 1. Let $r_\xi(x) = \pi_i(R)$, where $x = x_i$. Clearly, (ii) and (iii) hold. For the induction step, suppose range functions $r_\xi(y)$ have been defined for all variables y whose range-restriction is proven by a sub-proof of $i_1, ..., i_{n-1}$. We perform a case analysis on the rule i_n, which can be any rule of Definition 5.2.

1. same as the base case.

2. $r_\xi(x.i)(\mathrm{I}) = \pi_i(r_\xi(x)(\mathrm{I}))$.

3. $r_\xi(x)(\mathrm{I}) = \bowtie_{1 \leq i \leq n} \; r_\xi(x.i)(\mathrm{I})$.

4. $r_\xi(x)(\mathrm{I}) = r_\xi(y)(\mathrm{I})$ (in the case $x = y$), and
$r_\xi(x)(\mathrm{I}) = \{x \mid \exists t(x \in t \wedge t \in r_\xi(y)(\mathrm{I})\}$ (in the case $x \in y$), and
$r_\xi(x)(\mathrm{I}) = \{c\}$ (in the case $x = c$).

5. $r_\xi(x)(\mathrm{I}) = r_{\phi_i}(x)(\mathrm{I})$, where $x \in RR(\phi_i)$ ($i = 1$ or $i = 2$ depending on which is used in the proof).

6. $r_\xi(x)(\mathbf{I}) = r_{\phi_1}(x)(\mathbf{I}) \cup r_{\phi_2}(x)(\mathbf{I})$.

7. $r_\xi(x)(\mathbf{I}) = r_{\neg\phi'}(x)(\mathbf{I})$.

8. $r_\xi(x)(\mathbf{I}) = r_{\phi'}(x)(\mathbf{I})$.

9. $r_\xi(x)(\mathbf{I}) = \{\{y \mid I \models \phi'(y)\}\}$.

In cases 1-8, (ii) and (iii) follow immediately from the induction hypothesis. In case 9, Proposition 5.1 is used in addition to the induction hypothesis. It is needed to show that ϕ', a SAFE$_{LOGSPACE}$-CALC formula by the induction hypothesis, can be evaluated in LOGSPACE.

Consider now (b), i.e. RR-(CALC+*IFP*) $\subseteq$ SAFE$_{PTIME}$-(CALC+*IFP*). Proofs of range-restriction in RR-(CALC+*IFP*) are more complicated than those for CALC, since they involve computing the range-restricted columns of relations bound to *IFP* operators. Thus, the range functions are defined relative to a set of range-restricted columns for those relations. Let ξ be a RR-(CALC+*IFP*) formula and τ a mapping providing, for each non-database relation S, its range-restricted columns, computed as in Definition 5.3. The range function for a formula ξ, relative to τ, is denoted $r_{\xi,\tau}$. We proceed, as for CALC, by induction on the length of proofs showing range-restriction of variables. The rules used in the proof are those in Definition 5.3. The basis of the induction, and cases 1-8 in the induction step, are as for CALC, except that r_ξ is replaced by $r_{\xi,\tau}$. It remains to consider cases 1′, 9′ and 10.

1′. $r_{\xi,\tau}(x) = r_{\xi,\tau}(z_i)$, where x occurs in the i-th column of S and S is bound to the *IFP* operator $IFP(\phi(S), S)$ whose i-th free variable is z_i.

9′. $r_{\xi,\tau}(x) = \{IFP(\phi(S), S)\}$, and for each $y \in RR(IFP(\phi(S), S)(z_1, ..., z_n))$, $r_{\xi,\tau}(y) = r_{IFP(\phi(S),S)(z_1,...,z_n),\tau}(y)$. In this case, (ii) and (iii) follow immediately from the induction hypothesis and Proposition 5.1.

10. We need to compute the range function for each range-restricted variable corresponding to a range-restricted column of a formula $\xi = IFP(\phi(S), S)(z_1, ..., z_n)$. By definition, $RR_\tau(\xi) = RR_\tau(\phi)$. Recall that ϕ is iterated to compute the fixpoint. At the i-th iteration of ϕ, the ranges of variables depend on the ranges of the range-restricted columns of S at the $(i-1)$-st iteration. Thus, the definition of the range function follows the iteration of ϕ. We define the following sequence of range functions $r^i_{\phi,\tau}$ for each variable in $RR_\tau(\phi)$:

- $r^0_{\phi,\tau}(x)$ is computed as if S were a database relation, except that, in the base case (1), if $R = S$ then the value of the range function is $\emptyset$.
- $r^{i+1}_{\phi,\tau}(x)$ is computed as above, except that, in the base case (1), for $R = S$, $r^{i+1}_{\phi,\tau}(x) = r^i_{\phi,\tau}(z_j)$ if x occurs in the j-th column of S.

Note that, in the course of the iteration, $r^i_{\phi,\tau}(x)$ accumulates a number of ranges of variables from the 0-th iteration. More precisely, the sequence $\{r^i_{\phi,\tau}(x)\}_i$ is non-decreasing and, for each i, $r^i_{\phi,\tau}(x)$ is a union of $r^0_{\phi,\tau}(z)$ for some z among the variables in $RR_\tau(\phi)$. It follows that the iteration converges after a constant number of steps, say k. We define

$r_{\xi,\tau}(x) = r^k_{\phi,\tau}(x)$ for each x. It is easily seen that (ii) and (iii) hold. This completes the induction. □

Note that Theorem 5.1 extends to the languages with type restrictions :

Corollary 5.1 :

1. RR-CALC^k_i $\subseteq$ SAFE$_{LOGSPACE}$-CALC^k_i,
2. RR-(CALC^k_i+*IFP*) $\subseteq$ SAFE$_{PTIME}$-(CALC^k_i+*IFP*),
3. RR-(CALC^k_i+*PFP*) $\subseteq$ SAFE$_{PSPACE}$-(CALC^k_i+*PFP*).

The proof follows straightforwardly from the proof of Theorem 5.1.

We note that it is open whether the converse inclusions hold in Theorem 5.1 and Corollary 5.1. Consider for instance (2.). It may appear at first glance that the inclusion is strict, since there are range functions in PTIME which are not computable by RR-(CALC^k_i+*IFP*). However, a query which is safe wrt such a range function may also be safe wrt some other PTIME range function which *is* computable in RR-(CALC^k_i+*IFP*) (this was noted by Serge Abiteboul). Thus, the converse remains open.

We can obtain a partial converse of Corollary 5.1 in certain restricted cases. First, the converse holds with the assumption that an order $<_U$ on the atomic constants is given. Note that an order $<_U$ cannot be defined as in the proof of Theorem 4.1, since this cannot be done in a range-restricted fashion. Thus, $<_U$ has to be provided explicitly. For a language L, we denote by $L+ <_U$ the language augmented with an order relation $<_U$ which, on each input **I**, is assumed to hold a total order on $dom(U, atom(\mathbf{I}))$. The input **I** augmented with $<_U$ is said to be *ordered*.

Theorem 5.2 Let **R** be an $< i, k >$-database schema.

1. RR-(CALC^k_i + *IFP* + $<_U$) and SAFE$_{PTIME}$-(CALC^k_i + *IFP* + $<_U$) are equivalent and express precisely PTIME on ordered inputs over **R**.
2. RR-(CALC^k_i + *PFP* + $<_U$) and SAFE$_{PSPACE}$-(CALC^k_i + *PFP* + $<_U$) are equivalent and express precisely PSPACE on ordered inputs over **R**.

Proof : We only prove (1), since (2) is similar. The proof is based on the fact that all PTIME queries over ordered inputs can be expressed in RR-(CALC^k_i + *IFP*+ $<_U$). This follows from the proof of Theorem 4.1. Indeed, the proof provides, for each PTIME query q, a CALC^k_i + *IFP*+ $<_U$ formula $\phi_q(<_U)$ defining q. Furthermore, it is easy to check that the formula $\phi_q(<_U)$ constructed there is in fact range restricted, so is in RR-(CALC^k_i + *IFP*+ $<_U$). The difference with the proof of Theorem 4.1 is that there $<_U$ is postulated (in a non-range-restricted fashion) whereas here $<_U$ is assumed given.

It follows from Corollary 5.1 that RR-(CALC^k_i + *IFP* + $<_U$) $\subseteq$ SAFE$_{PTIME}$-(CALC^k_i + *IFP* + $<_U$), and from Proposition 5.1 that SAFE$_{PTIME}$-(CALC^k_i + *IFP* + $<_U$) $\subseteq$ PTIME. □

A second converse to Corollary 5.1 combines the density and range-restriction assumptions of Theorems 4.1 and 5.2. It no longer assumes that an order $<_U$ is provided. To allow the definition of $<_U$ in the language, the range-restriction assumption is relaxed for some non-trivial type T, and replaced by a density assumption for that type. Recall that a non-trivial type has set height at least 1 and tuple width at least 2. Clearly, a non-trivial type T allows representing an order $<_U$. To state the theorem, we need to define range-restriction relative to a set of types. If $\mathcal{T}$ is a set of types and L is a CALC-based language, $RR_{\mathcal{T}}$-L denotes the queries in L for which all variables of a type in $\mathcal{T}$ are range restricted.

Theorem 5.3 Let $\mathbf{R}$ be an $< i, k >$-database schema ($i \geq 1, k \geq 2$) and $\mathcal{I}$, $\mathcal{I} \subseteq$ **inst(R)**, be dense with respect to some non-trivial type T. Let $\mathcal{T}$ consist of all $< i, k >$-types other than T. Then :

1. $\mathrm{RR}_{\mathcal{T}}$-($\mathrm{CALC}_i^k$ + *IFP*)$|_{\mathcal{I}}$ and SAFE_{PTIME}-(CALC_i^k + *IFP*)$|_{\mathcal{I}}$ are equivalent and express precisely PTIME$|_{\mathcal{I}}$.
2. $\mathrm{RR}_{\mathcal{T}}$-($\mathrm{CALC}_i^k$ + *PFP*)$|_{\mathcal{I}}$ and SAFE_{PSPACE}-(CALC_i^k + *PFP*)$|_{\mathcal{I}}$ are equivalent and express precisely PSPACE$|_{\mathcal{I}}$.

Proof : We prove (1.). Consider the inclusions

$$\mathrm{RR}_{\mathcal{T}}\text{-}(\mathrm{CALC}_i^k + \mathit{IFP})|_{\mathcal{I}} \subseteq \mathrm{SAFE}_{PTIME}\text{-}(\mathrm{CALC}_i^k + \mathit{IFP})|_{\mathcal{I}} \subseteq \mathrm{PTIME}|_{\mathcal{I}}.$$

The second inclusion follows immediately from Proposition 5.1. Consider the first inclusion. For every query q in $\mathrm{RR}_{\mathcal{T}}$-($\mathrm{CALC}_i^k$ + *IFP*)$|_{\mathcal{I}}$, and every variable x of type T' ($T' \neq T$), there exists a PTIME range function $r_q(x)$. This is shown as in the proof of Theorem 5.1. We show that it also holds for variables of type T. For each variable x of type T, the range function is simply defined by $r_q(x)(\mathrm{I}) = dom(T, atom(\mathrm{I}))$. This is clearly in PTIME$|_{\mathcal{I}}$, since $\mathcal{I}$ is dense with respect to T.

The proof of the converse inclusions proceeds much like the proof in Theorem 5.2. The difference is that the order $<_U$ need not be provided. Indeed, $<_U$ can be defined in $\mathrm{RR}_{\mathcal{T}}$-($\mathrm{CALC}_i^k$ + *IFP*)$|_{\mathcal{I}}$. This can be easily done using an object of type T, since there are no range restrictions on type T. The proof of (2.) is identical, with *PFP* replacing *IFP* and PSPACE instead of PTIME. □

To conclude the section, we provide a result showing that sparsity assumptions can have an impact on the relative expressive power of range-restricted languages. Consider the language $\mathrm{CALC}_i = \cup_{k>0} \mathrm{CALC}_i^k$. We consider the connection between CALC_i and CALC_i + *IFP*. It has been proven in [GV90] that, in general, $\mathrm{CALC}_i \subset \mathrm{CALC}_i$ + *IFP*, by showing that the transitive closure of graphs with nodes of set height i is not expressible in CALC_i. The proof is based on an extension of Ehrenfeucht-Fraissé games to complex objects. The same proof implies the separation of RR-CALC_i and RR-(CALC_i + *IFP*). On the other hand, we show next that RR-CALC_i and RR-(CALC_i + *IFP*) are equivalent on sparse inputs. Thus, the *fixpoint* operator *IFP* provides no additional power in this case.

Proposition 5.2 Let $\mathbf{R}$ be an $< i, k >$-database schema and $\mathcal{I}$, $\mathcal{I} \subseteq$ **inst(R)** be sparse with respect to $< i, k >$-types. Then:

$$\text{RR-CALC}_i \mid_{\mathcal{I}} = \text{RR-}(\text{CALC}_i + \mathit{IFP}) \mid_{\mathcal{I}}.$$

Proof: The proof uses an encoding of objects of set height i by tuples of objects of lower set height. The encoding exists because the number of objects of set height i in the database, or relevant to the query, is bounded by a polynomial in the cardinality of the domains of lower types. Once the set height of objects has been reduced, fixpoints defining inductively relations over objects of set height $i-1$ can be simulated within CALC$_i$.

Let ϕ be a formula in RR-(CALC$_i$ + *IFP*). By Theorem 5.1, RR-(CALC$_i$ + *IFP*) $\subseteq$ SAFE$_{PTIME}$(CALC$_i$ + *IFP*). Hence, for each variable x in ϕ, there exists a range function $r_\phi(x)$ computable in PTIME wrt the size of the input I. Due to the sparsity of $\mathcal{I}$, there exists a fixed h such that, for $\mathrm{I} \in \mathcal{I}$, $\|\mathrm{I}\| \leq \|dom(i-1,k,D)\|^h$, where $D = atom(\mathrm{I})$. By Proposition 2.1, $\|dom(i-1,k,D)\| \leq |dom(i-1,k,D)|^j$ for some j. Thus, there exists m such that the number of objects of a type T of set height i occurring in the database or in the range of a variable in ϕ is bounded by $|dom(i-1,k,D)|^m$. Thus, all such objects of type T can be represented in a relation Q_T of arity $m+1$ and coordinates of a type S, of set height $i-1$, in the following way. For each such object o there exists an m-tuple $\vec{x}$ such that:

$$o = \{y : S \mid Q_T(\vec{x}, y)\}.$$

Clearly, the existence of such Q_T can be postulated using a variable of set height i. Given Q_T for each type T of set height i, all objects o of type T in an input instance $\mathrm{I} \in \mathcal{I}$ or in intermediate results can be represented using the corresponding m-tuple $\vec{x}$ provided by Q_T. Then, all relations defined inductively by an *IFP* operator involve objects of set height at most $i-1$, and any application of *IFP* can be expressed in CALC$_i$ alone. Finally, the result is obtained by performing a decoding where each m-tuple $\vec{x}$ representing an object of set height i is replaced with the corresponding object according to the appropriate Q_T. □

6 Restriction to flat queries

The complexity of query languages using complex objects has been previously evaluated with respect to the queries from flat databases to flat answers which they can express [HS88, HS89, KV88, PvG88]. In order to provide a connection with these results, we investigate in this section the expressive power of the languages introduced above with respect to the flat-to-flat queries they express.

Previous results on restrictions of CALC similar to CALC$_i^k$ provide complexity bounds but not exact expressiveness results [HS88, KV88]. It turns out that, again, fixpoint operators yield languages which have precise characterizations with respect to complexity. We obtain exact expressiveness results which emphasize, once again, the usefulness of the fixpoint operators *IFP* and *PFP*.

For a language L, let $(L)_0$ denote the queries in L whose input database schema and answer have types of set height zero. Consider the CALC$_i^k$ languages. From results of [HS88], it easily follows that $(\text{CALC}_i^k)_0 \subseteq \text{P}(hyper(i,k))$-time. Results from [GV90] imply that the inclusion is strict (see discussion preceeding Proposition 5.2).

The following shows that exact expressiveness results are obtained with the fixpoint operators.

Theorem 6.1 : Let $i \geq 1, k \geq 2$.

- $(\text{CALC}_i^k + \mathit{IFP})_0 = \text{P}(hyper(i,k))$-time,
- $(\text{CALC}_i^k + \mathit{PFP})_0 = \text{P}(hyper(i,k))$-space.

Proof: The result is a direct consequence of Theorem 4.2. Indeed, flat inputs are always dense with respect to $< 0, k >$-types and sparse with respect to all higher types. □

Recall that $\text{CALC}_i = \cup_{k>0} \text{CALC}_i^k$. The languages CALC_i were studied in [HS88], where upper complexity bounds were provided. More precisely, let $Hyper(i) = \{hyper(i,k) | k > 0\}$. It was shown in [HS88] that $(\text{CALC}_i)_0 \subseteq Hyper(i)$-time. Again, the strict inclusion follows from results in [GV90].

Results similar to those for CALC_i^k hold for CALC_i. The following is an immediate consequence of Theorem 6.1.

Theorem 6.2 : Let $i \geq 1$.

- $(\text{CALC}_i + \mathit{IFP})_0 = Hyper(i)$-time,
- $(\text{CALC}_i + \mathit{PFP})_0 = Hyper(i)$-space.

A natural question concerns the relationship between $(\text{CALC}_i)_0$ and $(\text{CALC}_i + \mathit{IFP})_0$. It was shown in [GV90] that $\text{CALC}_i \subset \text{CALC}_i + \mathit{IFP}$. However, this does not imply that $(\text{CALC}_i)_0 \subset (\text{CALC}_i + \mathit{IFP})_0$. This question remains open, and appears difficult. Indeed, for $i = 1$, the question reduces to that of whether the Polynomial Hierarchy $((\text{CALC}_1)_0)$ equals EXPTIME $((\text{CALC}_1 + \mathit{IFP})_0)$, which is an open problem. We note that the weaker inclusion $(\text{CALC}_i)_0 \subset (\text{CALC}_{i+1})_0$ was shown in [HS88], using a spectra theorem of Bennett [Ben62].

7 Conclusions

We investigated various ways of obtaining expressive and tractable query languages for complex object databases. Three main ideas emerge from the results:

- Fixpoint operators are shown to be useful constructs for queries on complex objects, and to yield languages well-behaved wrt expressiveness. This is due to the fact that they provide a tractable form of recursion, unlike the powerset operation.

- Type usage by the database has an impact on the complexity and expressiveness of calculus-based query languages. Type usage was formalized by the notions of density and sparsity wrt higher-order types. This can be thought of as information on the structure of the database, much like a form of integrity constraint.

- Syntactic conditions on queries can ensure tractability by limiting the range of variables to sets of objects computable from the database in a tractable fashion. Such conditions were provided in the form of range-restriction.

The results also point out that density and sparsity, and range-restriction, have similar effects and can be interchanged to some extent (Theorem 5.3). This provides a toolbox for tuning the complexity of queries by using a mixture of information on the database and restrictions on queries.

Among the expressiveness results, most notable is the characterization of PTIME under density assumptions (Theorem 4.1). Although this can be viewed essentially as a generalization of the known result that (flat) PTIME is captured by the *fixpoint* queries with order, the density assumption is a new assumption specific to complex objects, with no analog in the flat case. Sparsity is also shown to have an impact on complexity and on the relative power of languages (Proposition 5.2). An interesting problem left for future research is to consider the more realistic multi-sorted case where density and sparsity hold for some sorts and not others.

Our languages using fixpoint operators are closely related to known deductive languages with complex objects. Such deductive languages include LPS [Kup87], LDL [BNR+87] and COL [AG91]. Notions of range-restriction have also been defined for these languages, and the resulting languages are related to our range-restricted languages. Similar connections hold among the languages with restrictions placed on set height and tuple width, in the spirit of CALC_i^k. These connections are beyond the scope of this paper and are left for future work.

Acknowledgement: The authors wish to thank Serge Abiteboul for useful comments on a draft of this paper.

References

[AB86] S. Abiteboul and N. Bidoit. Non first normal form relations: an algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33(3):361–393, Dec. 1986.

[AB87] S. Abiteboul and C. Beeri. On the power of languages for the manipulation of complex objects. In *Proc. Int. Workshop on Theory and Applications of Nested Relations and Complex Objects (extended abstract)*, Darmstadt, 1987. INRIA research report n 846.

[AG91] S. Abiteboul and S. Grumbach. A rule-based language with function and sets. *ACM Transactions on Database Systems*, 16(1):1–30, March 1991.

[AH87] S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12:525–565, Dec. 1987.

[AK89] S. Abiteboul and P. Kanellakis. Object identity as a query language primitive. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 159–173, 1989.

[ASV90] S. Abiteboul, E. Simon, and V. Vianu. Non-deterministic languages to express deterministic transformations. In *Proc. 9th ACM Symp. on Principles of Database Systems*, 1990.

[AV89] S. Abiteboul and V. Vianu. Fixpoint extensions of first-order logic and datalog like languages. In *Proc. 4th Symp. on Logic in Computer Science*, pages 71–79, 1989.

[AV91] S. Abiteboul and V. Vianu. Non-determinism in logic-based languages. *Annals of Math. and AI*, 3:151–186, 1991.

[Ben62] J. H. Bennett. *On Spectra*. PhD thesis, Princeton University, 1962.

[BNR+87] C. Beeri, S. Naqvi, R. Ramakrishnan, O. Schmueli, and S. Tsur. Sets and Negation in a Logic Database Language (LDL1). In *Proc. 6th ACM Symp. on Principles of Database Systems*, pages 21–37, 1987.

[CH80] A. Chandra and D. Harel. Computable Queries for Relational Data Bases. *Journal of Computer and System Sciences*, 21(2):156–178, Oct. 1980.

[FT83] P. Fischer and S. Thomas. Operators for non-first-normal-form relations. In *Proc. 7th COMPSAC*, Chicago,1983.

[GS85] Y. Gurevich and S. Shelah. Fixed-point extensions of first order logic. In *Proc IEEE Foundations of Computer Science*, 1985.

[GV90] S. Grumbach and V. Vianu. Playing games with objects. In *Proc. Int. Conf. on Database Theory*, Paris, dec 1990.

[GV91a] S. Grumbach and V. Vianu. Expressiveness and complexity of restricted languages for complex objects. In *Proc. 3rd Int. Workshop on database programming languages*, Nafplion, Aug. 1991.

[GV91b] S. Grumbach and V. Vianu. Tractable query languages for complex object databases. In *Proc. 10th ACM Symp. on Principles of Database Systems*, pages 315–327, Boulder, May 1991.

[HS88] R. Hull and J. Su. On the expressive power of database queries with intermediate types. In *Proc. 7th ACM Symp. on Principles of Database Systems*, 1988.

[HS89] R. Hull and J. Su. Untyped Sets, Invention and Computable Queries. In *Proc. 8th ACM Symp. on Principles of Database Systems*, 1989.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Inf. and Control*, 68:86–104, 1986.

[Jac82] B. Jacobs. On Database Logic. *Journal of the ACM*, 29(2):310–332, 1982.

[KRS85] H.F. Korth, M.A. Roth, and A. Silberschatz. Extended Algebra and Calculus for not 1NF Relational Databases. Technical report, Department of Computer Science, University of Texas at Austin, 1985.

[Kup87] G.M. Kuper. Logic programming with sets. In *Proc. 6th ACM Symp. on Principles of Database Systems*, 1987.

[Kup88] G.M. Kuper. On the expressive power of logic Programming languages with Sets. In *Proc. 7th ACM Symp. on Principles of Database Systems*, 1988.

[KV84] G.M. Kuper and M.Y. Vardi. A new approach to database logic. In *Proc. 3rd ACM Symp. on Principles of Database Systems*, 1984.

[KV88] G.M. Kuper and M.Y. Vardi. On the complexity of queries in the logical data model. In *Proc. Int. Conf. on Database Theory*, pages 267–280, Bruges, 1988.

[PvG88] J. Paredaens and D. van Gucht. Possibilities and limitations of using flat operators in nested algebra expressions. In *Proc. 7th ACM Symp. on Principles of Database Systems*, pages 29–38, 1988.

[SS86] H. Schek and M. Scholl. the relational model with relation-valued attributes. *Information Systems*, 1986.

[Var82] M. Vardi. Relational queries computable in polynomial time. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.

[Ver86] J. Verso. *Verso: a Database Machine Based on Nested Relations*, volume LNCS 361, chapter Nested Relations and Complex Objects in Databases, pages 27–49. Springer Verlag, 1986.

ISSN 0249-6399